Notes on the Backpropogated Delta Rule algorithm.

1. What exactly are we trying to minimise?

As described, the learning rule implements gradient descent on

$$E = \sum_{p \in \mathrm{patterns}} \sum_{k \in \mathrm{outputunits}} (D_k^p - Y_k^p)^2$$

but other error measures are also valid.

For example

* different output units may have different importances

* the importance of each pattern may vary: e.g. it may be very important to get some patterns right.

* the use of the squared error may be inappropriate: one can use any even exponent, or the "*city block*" metric",

$$E = \sum_{p \in \mathrm{patterns}} \sum_{k \in \mathrm{outputunits}} \mid D_k^p - Y_k^p \mid$$

In each case, all that alters is the calculation of the error at the output units: the rest of the algorithm is unaffected.

There are are other measures, including some based in information theory: we will return to this topic later.

Termination.

When and how do we decide that either

* the network has converged
or
* the network is not going to converge?

This question also arises in the plain Delta rule.

We can choose to stop training the network when either

* the number of epochs exceeds some prespecified value
or
* the error is less than some prespecified value.

Note that for small enough h, the Delta rule will converge, and that the same is true for BP, simply because it is a gradient descent algorithm. However, for both algorithms, we cannot make guarantees about the size of the remaining error.

But which error?

If we choose the usual mean squared error, we run the danger of having almost correct output for almost all of the output units, or almost all of the patterns, but having a relatively large error for one unit, or for one pattern.

We can avoid this difficulty by using a different error measure: however,we can also use the usual error measure for training purposes, but alter the criterion we use for cessation of training. We can continue until

$$E_{\mathrm{maxerror}} = \max_p \max_k \mid D_k^p - Y_k^p \mid$$

(where p indexes the patterns, and k the output units) is less than some prespecified amount. E_{maxerror} gives the largest error on a single output unit for any pattern.

...and what value of error do we stop at?

If we are dealing with a classification network in which the desired outputs are 0 or 1, then we can use any E_{maxerror} such that $0 < E_{\mathrm{maxerror}}<0.5$. If however, we are aiming to produce particular output values, we will need to use a much smaller value.

The error surface.

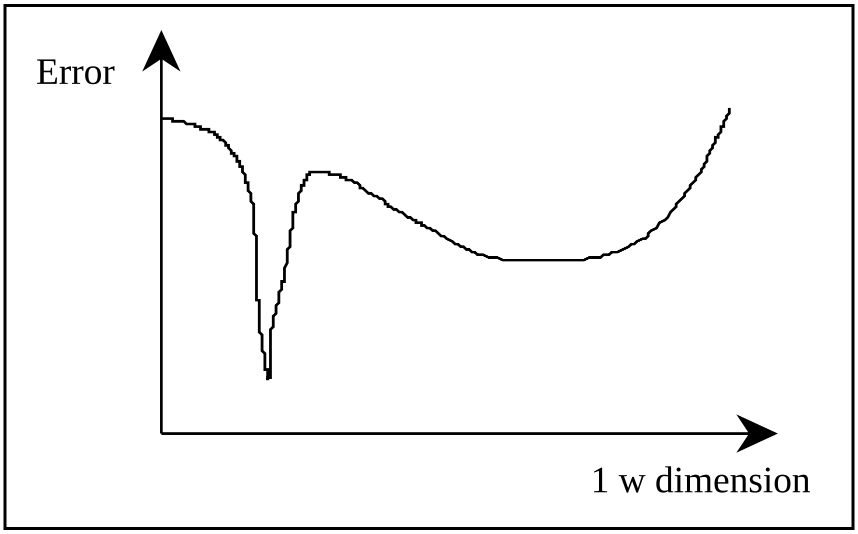
The error surface E (= E(**W**)) is a surface of high dimensionality (it is a function of all the weights), and (generally) very considerable complexity. Unlike the case with the simple Delta rule, it is not simply quadratic in each weight.

By the nature of the error measure, and of the output functions, E(**W**) is "well-behaved" - i.e. continuously differentiable, but unlike the case for the simple Delta rule,

E may have

* local minima
* ravines
* saddle points, etc.

These all present pitfalls.

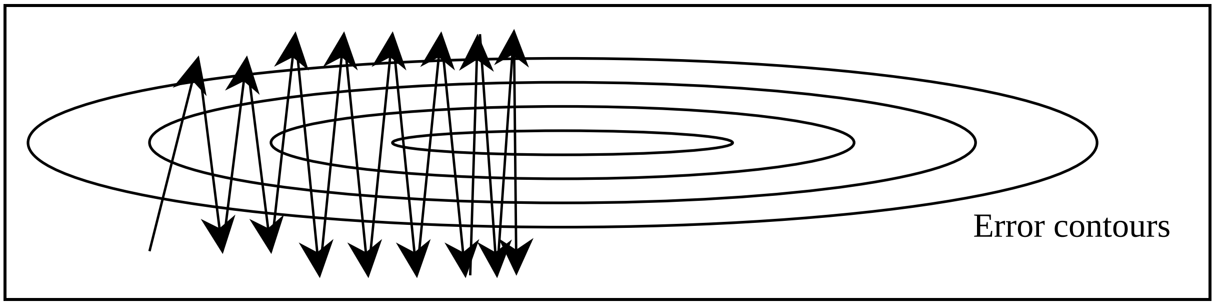


Here, the error has a sharp (global) minimum, and a smoother (local) minimum. It is very easy to miss the global minimum.

Speeding up convergence.

Although the proof of the gradient descent algorithm relies on the learning rate, η, being infinitesimally small, such η results in infinitely slow convergence. This is inconvenient.

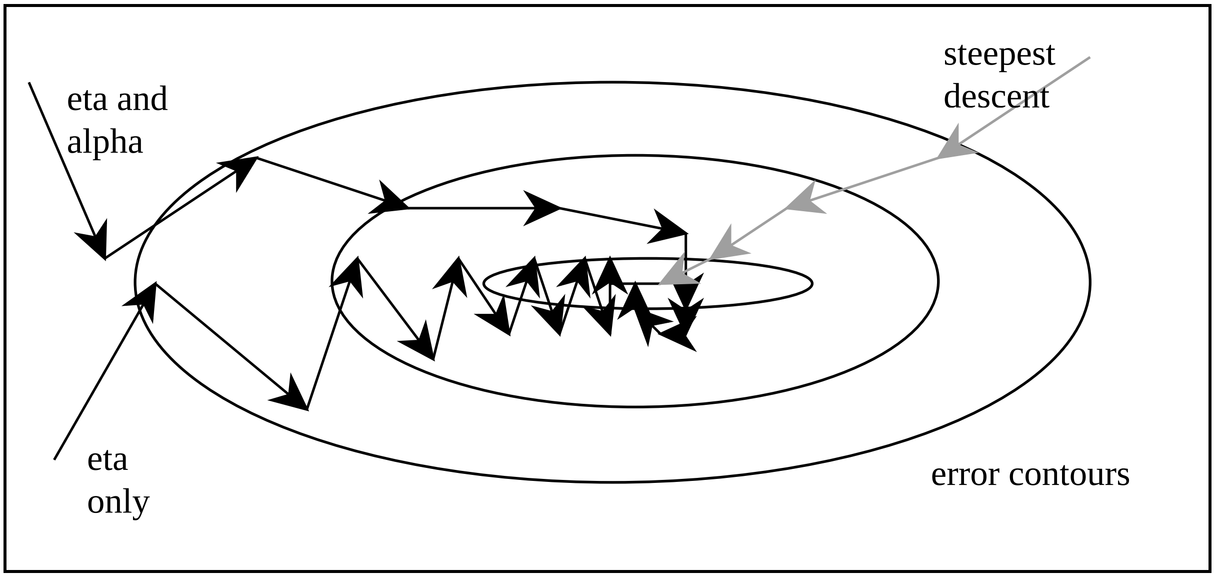
But if we use a value for η which is too large, we run the risk of missing minima because of overshoot. Also we have a risk of oscillation around a minimum.



One frequently chosen technique to overcome this is to keep η quite small, but to introduce a *momentum* term, α:

$$\Delta w_{ji}(t+1) = \eta E_j \frac{dY_j}{dA_j} Y_i + \alpha \Delta w_{ji}(t)$$

The effect of this is that $w_{ji}(t+1)$ tends to be in the same direction as $w_{ji}(t)$: indeed if a piece of error surface is of fixed steepness, the effect is to multiply η by 1/(1-α). The overall effect is to allow larger weight steps without introducing oscillation.



Note that the BP Delta rule does NOT implement steepest descent. All it does is to ensure that each Δw_{ji} is in the direction of gradient descent. The overall move in **W** will be such as to ensure decrease in E (assuming no overshoot in any direction), but is unlikely to implement steepest descent.

Steepest descent algorithms do exist (see Haykin p124, or Hertz Krogh and Palmer p125). These are non-local, and usually involve computation of 2nd derivatives.

What values should be used for η and α ?

How many layers should be used?

How many units should there be in each hidden layer?

One of the major problems with BP is that the answers to these questions are problem-dependent. But as a guide:

* generally, a value for η of between 0.03 and 0.1, and for α between 0.7 and 0.95 work reasonably well. But not always.

* use one hidden layer, at least until you are convinced that you need more than one.

* it is usually best to use a smaller number of hidden units than the number of input units, in order to allow a compact representation to form. But there can be other issues as well - and there is no good answer yet to this question.